VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

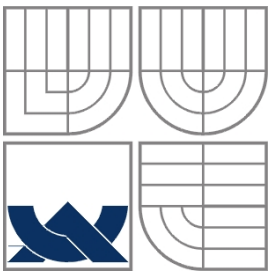
OBJEKTOVĚ – RELAČNÍ RÁMEC PRO PHP

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

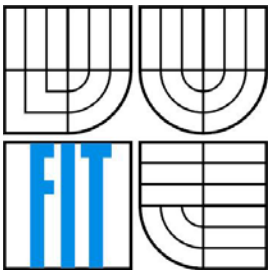
AUTOR PRÁCE
AUTHOR

Bc. MICHAL HUDEC

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

OBJEKTOVĚ – RELAČNÍ RÁMEC PRO PHP

OBJECT – RELATIONAL FRAMEWORK FOR PHP

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Michal Hudec

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Radek Burget, Ph.D.

BRNO 2007

Abstrakt

Cieľom tejto práce je navrhnúť a implementovať Objektovo-relačný rámec pre PHP. Tento rámec bude schopný mapovať objekty reálneho sveta do relačných databáz. V práci je navrhnutý vhodný spôsob špecifikácie metadát, ktoré popisujú uloženie dát do relačnej databáze. Samotný rámec je schopný ukladať, získavať a dotazovať sa objektových dát v relačnej databázy. Objektovo-relačný rámec je navrhnutý tak, aby umožňoval jednoduchú prenositeľnosť, medzi databázovými serverami.

Klíčová slova

Objektovo - relačný rámec, objektovo – relačný mapper, PHP, XML, metadáta, objektové databáze, relačné databáze, hibernate, JDO, ORR, mapovanie objektov, eval, extent, perzistentné objekty

Abstract

The objective of this work is to design and implement an Object-relational framework for PHP. This framework will be able to map objects to traditional relational database tables. In this work, an appropriate solution of metadata specification is presented. These metadata describe how an object can be store in a relational database. The framework itself is able to store, load and query any object data in relational database. This object-relational framework has been designed for simple portability among various database systems.

Keywords

Object-relational framework, object-relational mapping, PHP, XML, metadata, object oriented database, relational database, hibernate, JDO, ORR, object mapping, eval, extent, persistence objects

Citace

Michal Hudec: Objektovo – relačný rámec pre PHP, diplomová práce, Brno, FIT VUT v Brně, 2007

Objektovo – relačný rámec pre PHP

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Radka Burgeta, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Poděkování

Touto cestou by som chcel poďakovať môjmu vedúcemu diplomovej práce, pánovi Ing. Radku Brugetovi, Ph.D. za cenné rady a pripomienky k tejto práci.

Obsah

1 Úvod

Zmysel tejto práce spočíva v rozšírení skriptovacieho jazyka PHP o Objektovo – relačný rámec, ktorý nám umožní jednoducho a bez znalosti SQL príkazov pracovať s objektmi reálneho sveta a najmä nám umožní vkladať ich do relačných databáz.

V súčasnej dobe je výstavba webových aplikácií veľmi rozvinutá. Pri tvorbe aplikácií, ktoré pracujú s väčšími objemami dát, vždy skôr či neskôr narazíme na problém, akým spôsobom a kam dáta ukladať. Štandardom pre ukladanie dát sú v dnešnej dobe relačné databáze, ktoré túto úlohu plnia veľmi efektívne. Pri modelovaní reality sa však často dostaneme do situácie kedy musíme veľmi zložito ukladať objekty do relačnej databázy. PHP dospelo do fázy objektovo orientovaného jazyka. Je v ňom teda možné jednoducho tvoriť objekty reálneho života. Chýba nám však nástroj ktorý by umožnil ich jednoduché vkladanie do relačnej databázy.

V tejto práci sa zaoberáme práve vývojom daného nástroja. Popisujeme spôsob vytvorenia objektovo-relačného rámca (ORR) pre PHP. Jedná sa o knižnicu PHP, ktorá na základe určitého konfiguračného súboru mapuje atribúty objektov na domény v relačných tabuľkách.

V práci poukážeme na rozdiely medzi relačnými a objektovými databázovými serverami. Popíšeme existujúce objektové relačné rámce a na základe takto získaných poznatkov navrhneme vhodný spôsob metadát popisujúcich uloženie objektových dát v relačnej databáze.

Každá „tabuľka“ bude v metadatách popísaná ako určitá trieda. V PHP bude potom možné priamo vytvoriť objekt tejto triedy, ktorého atribúty budú odpovedať popisu v metadatách. Ukladanie týchto objektov bude možné pomocou metódy daného perzistentného objektu, ktorá vloží priamo celý objekt do databáze. Práca s objektmi uloženými v databázy bude teda možná aj bez znalosti SQL. Rámec však bude podporovať aj natívne príkazy SQL. Taktiež bude schopný vracať len určité položky objektov (napr.: názov), pre prípad, keď potrebujeme len efektívne využiť čiastkové dáta.

Práca je rozdelená do 7 kapitol. Nasledujúca kapitola nás oboznámi s relačnými a objektovými databázovými modelmi, popíšeme ich význačné rozdiely a taktiež sa oboznámime s tzv. objektovo-relačnými databázami, na ktorých vlastnostiach bude postavená táto práca.

Tretia kapitola všeobecne popisuje princípy objektovo–relačných rámcov. Dôvody zavádzania objektovo-relačného mapovania. Oboznámi nás so spôsobom akým je potreba objekty mapovať do relačných databáz. Taktiež obsahuje príklad malého objektové systému a návrh ako dáta tohto systému ukladať do relačných databáz.

Štvrtá kapitola popisuje existujúce objektovo – relačné rámce a to:

- JDO pre prácu s dátami v Jave.
- Hibernate

Piata sa zaoberá analýzou a návrhom nami vytváraného objektovo–relačného rámca pre PHP. Obsahuje analýzu vlastností, ktoré budeme od ORR očakávať. Popisuje formu metadát, ktoré

špecifikujú databázové schéma. Popisuje vzťahy, ktoré budú v ORR podporované. Taktiež si v tejto kapitole vysvetlíme princíp funkčnosti ORR. Uvedieme krátke príklady použitia priamo v určitej aplikácií.

V šiestej kapitole sa oboznámime so samotným spôsobom implementácie. Podrobne popíšeme spôsob, akým dochádza ku generovaniu kódu tried, ktoré sú špecifikované v metadátach. Uvedenie diagram tried reprezentujúci ORR. Detailne popíšeme jednotlivé metódy a vlastnosti týchto tried.

V siedmej kapitole uvedieme príklad použitia nami navrhnutého objektovo–relačného rámca v PHP pri vytváraní samotnej aplikácie. A zameriame sa najmä na porovnanie dotazovania na databázu v natívnom PHP a prácu s objektmi v našom ORR.

2 Relačné a objektovo orientované databáze

V tejto kapitole si predstavíme relačné, objektové a objektovo-relačné databáze. Popíšeme ich základné charakteristiky a ich principiálne rozdiely.

2.1 Relačné databáze

V relačnom modely dát je každá databáza zložená z tabuliek. Tabuľka je vyjadrením matematického pojmu *n-árna relácia* s prvkami vyjadrenými jednotlivými riadkami tabuľky. Záznamy v relačných databázach nie sú jednoznačne identifikované. Pokiaľ chceme na nejaký záznam odkazovať priamo identifikáciou, musí byť jeden zo stĺpcov tabuľky takzvaný identifikačný – *primárny kľúč*. [1]

Jedna zo základných nevýhod relačných databáz je, že neobsahujú vzťahy. Vzťah je totiž definovaný pomocou odkazu na identifikovaný záznam, avšak ako sme už spomenuli v relačných databázach nie sú záznamy jednoznačne identifikované. Z tohoto dôvodu sú vzťahy modelované na základe zhodnosti hodnôt v stĺpcoch tabuľky. Zväčša sa jedná o takzvaný *cudzí kľúč*, odkazujúci na *primárny kľúč* v inej tabuľke. V relačných databázach preto nie sú vzťahy uložené ako dáta, ale prakticky vznikajú až v priebehu dotazu na dané databázové schéma. Tento dotaz potom spojuje niekoľko tabuliek , čo môže byť výpočtovo náročné. Napríklad pre definovanie vzťahov typu M:N je potreba spojovacia (väzbová) tabuľka. [1]

Samotná realizácia týchto „vzťahov“ v relačných databázach súvisí so schopnosťou vyhľadávať v tabuľkách podľa obsahu. Vzhľadom k tomu dochádza k transformácií pamätí s priamym prístupom na asociatívne vďaka tzv. *indexovým súborom*. Rýchlosť tohoto prevodu je jedným zo základným príznakov efektivity relačných databáz. [1]

Výhoda relačných databáz spočíva najmä v tom, že sú relatívne jednoduché a ľahko pochopiteľné. Navyše na bežné aplikácie kancelárskeho typu plne postačujú. Relačných databáz je veľmi veľa a od rôznych firiem. Takmer každá ponúka určité vylepšenia, ale v princípe dodržujú štandard dotazovacieho jazyka SQL.

2.2 Objektovo orientované databázy

Človek inklinuje k pozorovaniu svojho okolia ako množiny objektov. Tieto objekty majú svoje vlastnosti, svoje špecifické chovanie, sú vo vzťahu s inými objektmi, modelujú realitu.

Objektovo orientovaný model databáz sa snaží zachovať spomínané vnímanie sveta a dá sa teda povedať, že sa jedná o množinu vzájomne sa ovplyvňujúcich objektov. Táto vlastnosť robí síce daný systém ľahko pochopiteľný z hľadiska komplexnosti, avšak samotná tvorba štruktúry objektov v databázy je zložitejšia ako v relačných databázach. Odmenou nám ale zase sú jednoduchšie dotazy na databázu.

Objekt je určitá entita, ktorá reprezentuje istý objekt z reality, a ktorá si je schopná uchovať určitú stavovú informáciu. Nad objektom je možné vykonávať určitú množinu operácií, ktoré následne menia jeho stav (stavovú informáciu). [1]

Každý objekt si svoje informácie skrýva – informácie sú zapuzdrené do daného objektu, to znamená, že s danými vlastnosťami objektu je možné manipulovať jedine využitím operácii, ktoré objekt poskytuje. Tento spôsob prístupu k informáciám o objekte znamená, že nemusíme poznať formu ani spôsob, ako boli vlastnosti objektu implementované, stačí poznať dané operácie, ktoré k nim pristupujú. Táto vlastnosť u objektov sa nazýva *polymorfizmus* [1]

Objekty zhodných vlastností sú organizované do tried objektov. Objekty rovnakej triedy sa označujú ako inštancie danej triedy. Triedy objektov sú organizované podľa vzťahu dedičnosti medzi objektmi rovnakým spôsobom, ako v objektových programovacích jazykoch. [4]

Každý objekt má v rámci celej svojej pôsobnosti jednoznačnú identifikáciu – *OID (object identifier)*. Tento identifikátor umožňuje tvorbu *trvalých vzťahov*. Aj v objektovom modely sa objavuje problém vyhľadávania podľa obsahu, avšak v tomto modely nám vždy postačí jediný typ, podľa ktorého sa vyhľadáva – OID. Vzhľadom k existencií konceptu OID môžeme rozlišovať medzi pojmom rovnosť a totožnosť objektu. Dva objekty so zhodnými dátami, ešte nemusia byť totožné. Všimnime si rozdiel oproti relačnému modelu, kde by rovnaké hodnoty predstavovali redundanciu dát.

S objektmi sa v databáze komunikuje pomocou posielania správ, tak ako v objektových programovacích jazykoch. Každá správa predstavuje žiadosť o určitú operáciu nad objektom, ktorému sa správa posiela. Objekt reaguje vykonaním odpovedajúcej metódy. V správe je možné prenášať dáta (aj objekty), ktoré následne sú využité vo volaných metódach – dátový tok v smere posielania správ. Ak objekt na základe správy vykoná určitú metódu, tak môže dáta aj vydať – dátový tok proti smeru posielania správy. [4]

Objektová databáza podporuje viacej typov množín objektov.

2.3 Vybrané vlastnosti objektových a relačných databázových modelov

Objektový a relačný databázový model sa výrazne odlišujú. Uvedieme si teda ich hlavné rozdiely:

- *Objektový model* podporuje teoreticky neobmedzené zložité a štruktúrované objekty, ktoré tým pádom jednoduchšie korešpondujú s reálnym objektom v zadaní. Je možné vytvárať objekt ako kombináciu existujúcich, pričom navonok sa objekt tvári ako jedna celistvá entita. Tieto vlastnosti sa kladne prejavujú najmä na dotazovacích možnostiach aplikácie.
  Relačný model je obmedzený len na záznamy jednoduchých dátových typov, ako sú čísla, texty. Každý zložitejší údaj musí byť transformovaný na sústavu viacerých tabuliek, čo pri dotazovaní výrazne zaťažuje výpočtový modul.
- *Relačný model* má vzťahy uložené v databázach len staticky – vo forme hodnoty. Dynamická stránka musí byť implementovaná programom, ktorý spolupracuje s databázou.
  Objektový modul pracuje s objektmi, ktoré majú v databáze uložené aj kódy metód, takže dynamická stránka manipulujúca s objektmi je nezávislá na obslužnej aplikácií. To znamená, že objektová databáza môže byť navrhnutá tak, že nepotrebuje obslužnú aplikáciu, jej funkčnosť zaistia samotné objekty a ich metódy.
- *Relačný model* umožňuje priamy prístup k údajom záznamu cez príslušné domény.
  Objektový model svoje údaje skrýva a nedá sa k nim pristúpiť priamo, ale len pomocou vybraných správ, ktoré aktivujú dané metódy. Medzi hodnotou vlastnosti objektu a metódou objektu nemusí byť priama súvislosť, ale daná metóda môže vracať výsledok ľubovolného výpočtu.
- *U relačného modelu* je nutné, aby všetky údaje v rámci jednej domény zachovávali rovnaký dátový typ a všetky záznamy jednej tabuľky rovnakú štruktúru. Množiny skladajúce sa s rôznych prvkov je potreba rozkladať. Vzťahy je potreba realizovať pomocou prepojenia viacerých tabuliek. To má za následok užívateľsky a časovo náročné operácie pri dotazovaní sa na zložitejšie štruktúry.
  V objektovom modely dochádza k spojovaniu len výnimočne, pretože možnosti skladania objektov sú prakticky neobmedzené.
- *V relačnom modely* dochádza pri dotazovaní na zložitejšiu dátovú štruktúru k častému spojovaniu viacerých tabuliek

Relačný model	Objektový model
neexistuje jednoznačná identifikácia	jednoznačné OID pre každý objekt
vzťahy vyjadrené pomocou cudzích klúčov	vzťahy sú vytvárané pomocou referencií na OID
pre vzťahy M:N je nutné vytvoriť spojovaciu tabuľku	atributy objektu môžu byť aj iné objekty
	vzťahy M:M je možné vytvárať priamo

Tabuľka č.1 – vybrané rozdiely medzi relačným a objektovým modelom dát

2.4 Objektovo – relačné databáze

Po pojmom objektovo – relačné databázové systémy si predstavíme relačné databázové systémy s objektmi, alebo relačné databáze, ktoré rôznymi spôsobmi prekračujú, alebo rozširujú relační dátový model. V praxi sa jedná o množstvo systémov s rôznymi vlastnosťami. Často dochádza aj k ich kombinácií. Tieto systémy si teraz predstavíme.

2.4.1 NFNF databáza

Non first normal form - sú to databázové systémy, ktoré nepožadujú normalizáciu do 1NF. Z toho vyplýva, že v týchto databázach je možné do jednej položky relačnej tabuľky vkladať viac ako jednu hodnotu. Táto hodnota môže byť buď určitý, užívateľom definovaný štruktúrovaný dátový typ, alebo celá relácia, čo znamená, že položkami tabuliek sú tabuľky – táto varianta sa nazýva *nested relations.*

2.4.2 Relačná databáza, umožňujúca vkladať objekty do tabuľky

Jedná sa o objekty vytvorené v objektovom programovacom jazyku. Samotný systém riadenia báze dát (SRBD) je samozrejme relačný a prevádza len operácie zápisu a čítania v relačných tabuľkách. Operácie typické pre objekty sa prevádzajú v danom programovacom jazyku. Pre tento systém sú charakteristické nasledovné obmedzenia:

- Do relačných tabuliek sa neukladajú metódy objektov, ale len dáta objektov. Metódy zostávajú len v danej aplikácií.
- Objekty sa buď ukladajú do jednej vyhradenej domény tabuľky, alebo sa rozkladajú do viacerých domén jednej tabuľky. Pri obidvoch prípadoch ale musí platiť pravidlo relačných databáz a to, že v jednej doméne musia byť dáta len jedného typu. Keď zoberieme do úvahy že sa ukladá buď celý objekt alebo jeho vlastnosti, tak môžeme tvrdiť, že v jednej tabuľke môžu byť uložené objekty len jedného typu. Inými slovami tabuľka odpovedá triede objektov.

2.4.3 Relačná databáza umožňujúca ukladať dynamické typy dát – označované ako triggers

Trigger je malá časť kódu, ktorým sa definujú dátové závislosti medzi hodnotami v tabuľkách. Každý trigger musí mať:

- Definovanú aspoň jednu doménu v tabuľke, ktorej dáta sleduje.
- Definovanú udalosť nad sledovanými dátami, ktorou sa spúšťa jeho činnosť.
- Algoritmus aspoň nad jednou doménou v tabuľke, ktorý popisuje jeho činnosť.

Pomocou triggeru je možné napríklad zabezpečiť, že pri vložený nejakej hodnoty na sledovanú doménu dôjde k automatickému výpočtu a uložení jeho hodnoty do inej domény.

2.4.4 Vypočítateľné atribúty

Predstavujú ďalší dynamický typ dát v relačných databázach. Tak ako v predchádzajúcom systéme aj teraz sa jedná o malé časti kódu nad doménami v tabuľke, ktoré však tentoraz neslúžia k definovaniu ďalších domén tabuľky, ich výsledok so nikam neukladá. Princíp vypočitateľných atribútov je analogický k správam parametrov, ktoré po poslaní príslušnému objektu vracajú hodnoty určené výpočtom z dátových zložiek tohto objektu. Vypočítateľné atribúty sú výrazným skokom smerom k objektovému modelu.

3 Objektové relačné rámce

Objektové relačné rámce (ORR) predstavujú prostredie, ktoré poskytuje prostriedky potrebné k objektovo relačnému mapovaniu.

3.1 Objektové relačné mapovanie

Objektové relačné mapovanie (ORM), konvertuje dáta v tabuľkách relačnej databázy na objekty a naopak umožňuje aj uloženie objektov do relačných databáz. ORM slúži k tomu aby bolo možné jednoducho používať relačné databáze v prostredí objektovo orientovaných programovacích jazykov. Perzistentné objekty sú potom priamo vo forme dát uložené do databáze. Výsledkom je efekt virtuálnej objektovej databáze. Dáta, ktoré sú vo forme objektového návrhu, sa však nedajú jednoducho previesť do relačnej databáze a opačne. Preto je potreba zaviesť formu mapovania. Mapovanie slúži k načítaniu jednotlivých dát z relačnej databáze a následnému naplneniu príslušných dátové položiek objektov a opačne k uloženiu dátových položiek objektov do jednotlivých domén daných tabuliek v relačnej databázy. Primárna snaha mapovania smeruje k čo najlepšiemu využitiu OOP (objekty reprezentujú objekty reálneho sveta) a možnosti relačnej databáze (indexy, primárne kľúče, atď.).

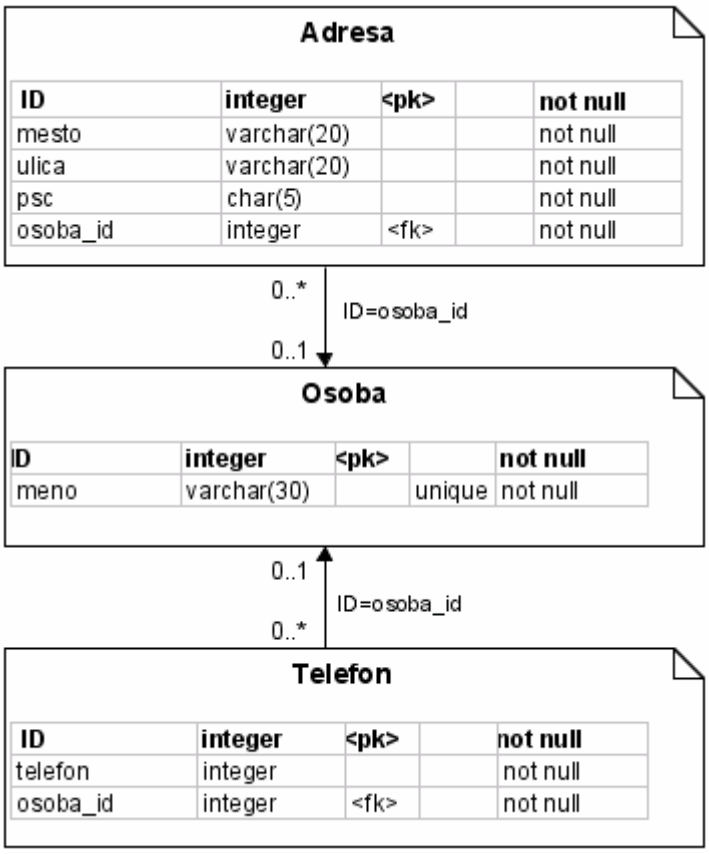
3.2 Princíp mapovania

Objekty samotné sú často zložité dátové štruktúry. Môžu napríklad obsahovať ďalšie objekty alebo iné „neskalárne“ hodnoty.

Predstavme si napríklad položku v adresári, kde záznam osoba má žiadne alebo viac telefónnych čísel a žiadnu alebo viac adries. V objektovo orientovanej implementácií by túto položku modelovali objektom *osoba* s vlastnosťami napríklad: *meno*, *zoznam telefónnych čísel a zoznam adries.* Zoznam telefónnych čísel by sám obsahoval inštancie triedy – *telefónne číslo*. Zoznam adries obdobne inštancie triedy *adresa.* Pri uložení objektu osoba do objektovej databáze je všetko v poriadku. Avšak pri ukladaní do relačnej databázy musíme túto štruktúru rozložiť na 3 tabuľky – *osoba, telefónne číslo a adresa.* Pričom triedy telefónne číslo a adresa musia obsahovať odkaz (tzv. *cudzí kľúč*) na tabuľku osoba [viď kap. 2.1], pomocou ktorého zabezpečíme prepojenie záznamov. Uvedený príklad nám bližšie znázorňujú nasledujúce diagramy.

Na *obrázku č. 1* sa nachádza konceptuálny diagram tried znázorňujúci model položky v adresári. *Obrázok č. 2* znázorňuje logické schéma databáze v relačnom modely dát. Zatiaľ čo v objektovej databáze by sme uložili objekt položka z adresára, ktorý by bol zložená štruktúra ako jednu celistvú entitu, v relačnej databáze je potreba vložiť záznam do každej z tabuliek.

Obr. č. 1 – konceptuálny diagram tried pre položku v adresári



Obr. č. 2 – Logické schéma databáze (položka adresára)

Ako je z logickej schémy databáze patrné jedna tabuľka predstavuje jednu triedu. Jednotlivé riadky tabuľky potom predstavujú inštancie triedy (objekty). Samozrejme sa jedná o veľmi jednoduchý príklad, pre ktorý by samotný ORR význam nemal, ale v praxi sa často stretávame s nutnosťou modelovať systém ako množinu reálnych objektov.

3.3 Ďalšie vlastnosti O-R rámca

Vyššie spomenutý spôsob mapovania objektov je základná vlastnosť ORR, ktorú musí plniť z hľadiska princípu. Avšak ORR by mal taktiež riešiť problematiku dedičnosti, mal by byť schopný využiť silu relačných databáz vo vyhľadávaní a filtrovaní údajov. ORR by mal taktiež riešiť efektívny prenos dát, napr.: pokiaľ chcem získať len zoznam objektov do číselníka, postačí ID objektu a jeden parameter textovej podoby, nie je teda vždy nutné získavať všetky údaje z databázy, preto by mal byť ORR schopný vytvárať zástupné objekty, ktoré budú predstavovať oklieštenú verziu objektu. [7]

ORR teda prináša veľa výhod, ale aj niektoré nevýhody, napríklad pri jeho používaní v malých aplikáciach, môže byť skôr spomaľujúci prvok ako urýchľovací

Výhody ORR:

- rýchle a spoľahlivé ukladanie veľkého objemu dát v relačných databázach
- čisto objektový prístup k programovaniu, v zdrojovom kóde nie je množstvo SQL dotazov

Nevýhody ORR:

- môže byť pomalší ako natívna implementácia, pretože sa jedná o mapovanie
- niektoré objekty reality môže byť ťažko prevoditeľné do relačnej DB

4 Súčasné objektovo relačné rámce

V súčasnej dobe sa pi vývoji podnikových aplikácií najčastejšie používa objektovo orientovaný prístup, zatiaľ čo štandardom pre ukladanie dát zostávajú relačné databáze. Pri vývoji týchto aplikácií sa najviac času venuje zväčša písaniu kódu, ktorý má za úlohu udržovať perzistentné dáta. Preto je mimoriadne dôležité oddeliť vlastnú logiku aplikácie od konkrétneho spôsobu uloženia dát. Preto vznikajú rôzne rámce, ktoré umožňujú mapovanie dát medzi objektovo orientovaným svetom a relačnými databázami. Používanie týchto rámcov uľahčuje prácu programátorom tým, že sa nemusia zaoberať konkrétnym databázovým modelom, ale sústrediť sa len na logiku aplikácie. [8]

V nasledujúcich častiach správy si detailnejšie popíšeme niektoré známe existujúce objektovo relačné rámce.

4.1 Java Data Objects

Java Data Objects (JDO) je štandardizované rozhranie pre perzistenciu objektov v Jave. JDO umožňuje implementovať ľubovoľný spôsob ukladania objektov. Na jednej strane môže slúžiť ako rozhranie objektovej databáze, na strane druhej môže ukladať dáta objektov do relačnej databáze - mapovať objekty. Z technického hľadiska sa jedná o abstraktnú vrstvu medzi objektmi Javy aplikácie a skladiskom perzistentných dát. JDO sa vyskytuje vo viacerých implementáciach. Ich jednotlivá úroveň však je veľmi rozdielna a tieto implementácie často ani nie sú navzájom kompatibilné. V súčasnej dobe prevláda podpora relačných databáz. My sa ďalej budeme zaoberať prevažne implementáciou *Sun JDO* od firmy Sun Microsystems, ktorá ja asi najvýznamnejšia. Medzi ďalšie implementácie patrí napríklad *Kodo JDO, LiDo, JDOGenie, Castor JDO, ObjectSore*. U niektorých spomenieme ich hlavné vlastnosti.

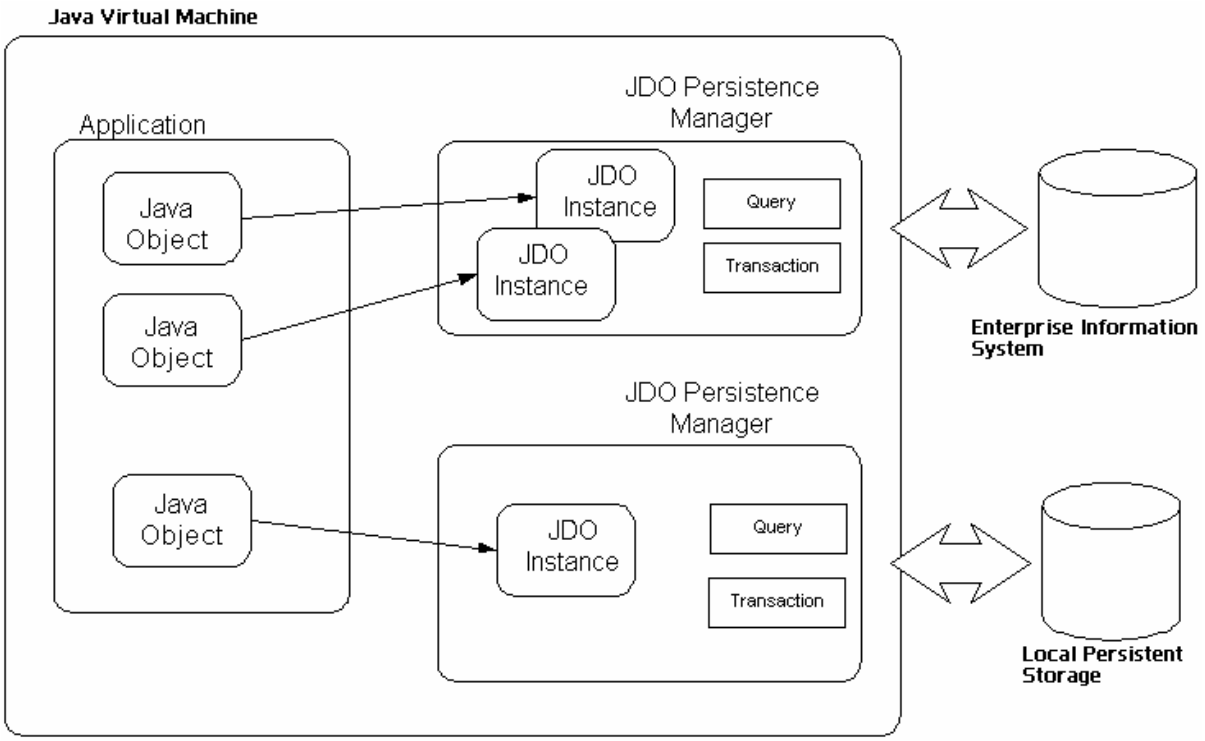
Ako sme už spomenuli, tak JDO nám umožňuje plne objektový prístup a nie je teda nutné mapovať objekty v kóde aplikácie na konkrétne dátové typy pre daný typ databáze. Samozrejme, že je mapovanie aplikačných objektov na objekty, resp. relácie musí byť poskytnuté aj v JDO. Avšak v JDO je mapovanie celkovo oddelené od kódu aplikácie. Mapovanie prebieha na základe externých XML súborov. JDO umožňuje taktiež transakčné spracovanie dát. [11]

4.1.1 Sun JDO

Základnou vlastnosťou objektového modulu je transparentnosť prístupu k dátam. Takmer vo všetkých prípadoch je možné pristupovať ku všetkým objektom, ktoré sú uložené v databázy. Pokiaľ však zoberieme do úvahy hľadisko efektivity, tak je veľmi dôležité aby aplikácia v jednom okamžiku pracovala s čo najmenším počtom objektov z databázového zdroja naraz. Práve toto je úloha JDO.

Aby vytváral ilúziu, že aplikácia pristupuje k všetkým objektom naraz, zatiaľ čo v skutočnosti pristupuje len k malej podmnožine objektov, ktoré sú skutočne inicializované v JVM. [11]

Nasledujúci obrázok ilustruje architektúru Sun JDO, ktorá umožňuje aplikácia pripájať sa k viacerým dátovým zdrojom.



Obr. č. 3 – Architektúra Sun JDO [10]

Na obrázku je zobrazený dvojaký spôsob použitia JDO v praxi. V prvom prípade ide o pripojenie „Enterprise Information System“, ktoré reprezentuje pripojenie typu klient/server ku zdieľanému zdroju dát. Druhý prípad ukazuje možnosť využitia JDO pre prístup k dátam uloženým v lokálnom úložisku.

Na obrázku vidíme základné prvky JDO. V rámci jedného JVM je spustená aplikácia, ktorá obsahuje objekty s dátami. Tieto objekty sú inštancie JDO objektov. Pracuje s nimi a s databázou komunikuje PersistenceManager. Ten taktiež umožňuje získavať objekty z databázy pomocou dotazov. JDO využíva nasledujúce rozhrania *PersistentCapable, PersistenceManager, PersistenceManagerFactory, Transaction, Query. [10]*

- **PersistentCapable –** je to akýsi základný objekt celej JDO technológie. Je to rozhranie, ktoré umožňuje aby daný objekt bol persistentný. Každý objekt, ktorý má byť persistentný musí byť odvodený od triedy, ktorá implementuje toto rozhranie.
- **PersistenceManager** – ako sme už spomenuli, je to rozhranie pre JDO inštancie, zabezpečuje riadenie transakcií, alebo ukladanie dočasných dát. Vytvára taktiež rozhranie *Query*

- **PersistenceManagerFactory –** vytvára inštancie rozhrania PersistenceManager a umožňuje nastaviť chovanie transakcií či pripojenia.
- **Transaction –** poskytuje metódy pre nastavenie chovania transakcií a u samostatne bežiacich aplikácií zabezpečuje úspešné vykonanie transakcie v databázy.
- **Query –** je vytvárané a spúšťané v rámci inštancie PersistenceManager a pre dotazy do databáze používa Object Query Language (OQL).

4.1.2 Konfigurácia

Skôr ako môžeme začať používať ľubovoľnú implementáciu JDO, je potreba ju nakonfigurovať. Je potreba špecifikovať príslušný databázový server, s ktorým sa bude pracovať, príslušnú databázu a meno a heslo, pomocou ktorých budeme k serveru pristupovať. Tieto nastavenia sa vykonávajú pomocou javovských *properties.* Tieto hodnoty nemusia byť špecifikované v aplikácií, ale je možné ich nastaviť v externom súbore. Podrobnejšie je daná problematika spracovaná v literatúre [13].

4.1.3 Metadáta tried v Sun JDO

Pre triedy, ktorých objekty majú byť perzistentné, je potreba špecifikovať informácie o spôsobe ich uloženia v databázy. Špecifikácia metadát sa tvorí pre každý balík tried (adresár s triedami perzistentných objektov) a je uložená súbore package.jdo v rovnakom adresári. Špecifikácia využíva formát XML. Predstavme si jednoduchý príklad triedy *Osoba*: [13]

```
Package org.people
public class Osoba
{
    protected String jmeno;
    protected String prijmeni;
    protected int vek;
    public Osoba() { }
    public Osoba(String jmeno, String prijmeni, int vek)
    {
      ...
    }
}
```

Metadáta pre túto triedu uložíme do rovnaného adresára do súboru *package.jdo* a špecifikujeme následovne:

```
<jdo> <package name="org.people">
<class name="Osoba" identity-type="datastore">
    <field name="jmeno">
       <extension vendor-name="jpox" key="length" value="max 100"/>
    </field>
    <field name="prijmeni">
        <extension vendor-name="jpox" key="length" value="max 100"/>
    </field>
    <field name="vek"/>
</class>
</package> </jdo>
```

Každá trieda je v metadátach špecifikovaná značkou <*class*>. V uvedenom príklade definujeme meno triedy a že OID si generuje samotné JDO. Musíme taktiež zadať všetky vlastnosti triedy, ktoré majú byť ukladané do databáze, a to pomocou značky <*field*>. Dátový typ každého atribútu vyplýva z deklarácie triedy, takže v metadátach ho nie je potreba uvádzať.

4.1.4 Spôsob použitia JDO

- Na začiatku existuje ľubovoľná Java trieda. U tejto triedy požadujeme schopnosť perzistencie
- Po kompilácií triedy je na ňu aplikovaný špeciálny proces rozšírenia, na ktorého konci, začne trieda implementovať interface *PersistentCapable*, ktorý je súčasťou špecifikácie JDO. Modifikácia triedy je prevedená až na úrovní byte kódu, preto pôvodný zdrojový kód zostane nezmenený
- K tomu aby prebehol predchádzajúci bod je potreba vytvoriť pre každú triedu *persistence descriptor* (vo formáte XML), ktorý obsahuje konfiguráciu perzistencie, napr.: naviazanie atribútov triedy na stĺpec v databázy
- Teraz už je možno triedu definovaným spôsobom mapovať (ukladať / nahrávať) do perzistentného skladu

4.1.5 Transakcie

Práca v JDO s perzistentnými objektmi prebieha v rámci *transakcií*. Transakcia je postupnosť operácií s perzistentným objektom, buď prebehne celá alebo neprebehne vôbec (ak niektorá z operácií zlyhá). Vďaka transakciám sa aplikácia vždy nachádza v konzistentnom stave. [17]

Ku každému vytvorenému objektu triedy *PersistenceManager*, ktorý je vytváraný jeden pre celú aplikáciu a poskytuje rozhranie pre riadenie perzistencie objektov, je automaticky vytvorený jeden objekt triedy *Transaction*, ktorý je prístupný pomocou metódy *currentTransaction()*. Pomocou metód takto získaného objektu môžeme riadiť priebeh transakcií. Ako príklad si uvedieme vytvorenie nového perzistentného objektu:

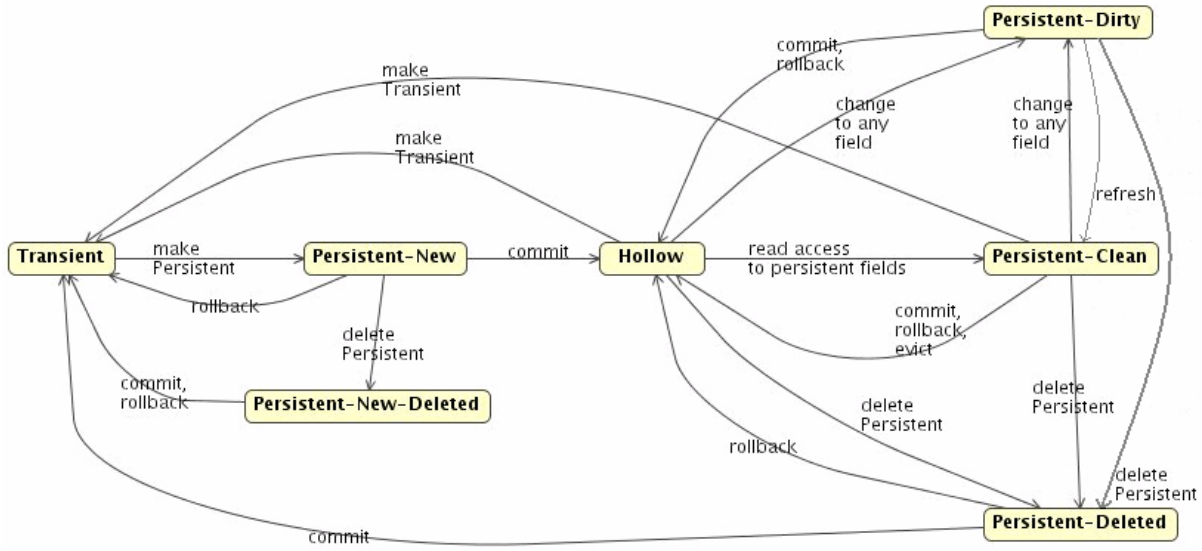
```
Transaction tx = pm.currentTransaction();
try {
    tx.begin();
    Osoba osoba = new Osoba("Michal", "Hudec", 55);
    pm.makePersistent(osoba);
    tx.commit();
}
finally
{     if (tx.isActive())   {  tx.rollback();  }     }
```

Stav vytváraného objektu *Osoba* je zobrazený na obrázku *č. 4*. Bližšie informácie o stavoch objektov, do ktorých sa v rámci svojho života môžu dostať, popíšeme v nasledujúcej kapitole.

Obr. č. 4 – stavy ktorými prejde objekt pri vytváraní a následnom uložení [17]

4.1.6 Životný cyklus JDO inštancií

Každá inštancia JDO má svoj životný cyklus. To znamená, že za behu aplikácie sa môže nachádzať vždy v niektorom z rôznych stavov, medzi ktorými môže prechádzať. Tieto stavy reprezentujú stav objektu v bežiacej aplikácií. Prechod medzi stavmi objektu môže nastať volaním metód objektu PersistenceManager, zmenou stavu prebiehajúcej transakcie, alebo zmenou dát objektu. V priebehu akejkoľvek transakcie môže objekt prejsť nasledujúcimi stavmi: *Transient, Persistent a Hollow*. Stav persistent sa ďalej delí na podstavy *Persistent New, Persistent Clean, Persistent Dirty, Peristen Deleted, Persistent New Deleted.* Spomínané stavy sú znázornené na obrázku č. 5



Obrázok č. 5 – stavy JDO inštancie [12]

- **Transient** – objekt bol vytvorený za behu aplikácie. V tomto stave sa nachádza každý nový objekt. (po volaní *new*). Persistentný objekt sa stane transientným po volaní jednej z metód *makeTransient*, alebo *deletePersistent*. O objekty v tomto stave sa JDO nijak nestará.
- **Persistent** – persistentný objekt. Objekt je pod správou JDO. Tento stav sa delí na rôzne podstavy na základe toho aký je vzťah dát v inštancií objektu a v dát uložených v sklade objektov.
- **Hollow** – objekt reprezentujúci dáta v sklade objektov, ale hodnoty týchto dát nie sú súčasťou objektu v pamäti. Persistentný objekt sa do tohoto stavu dostane ukončením transakcie (po volaní *commit*). Nachádzajú sa tu všetky novo načítané objekty zo skladu. Prečítaním, alebo zmenou dát objektu dochádza k načítaniu dát zo skladu do objektu v pamäti a objekt tým pádom prechádza do stavu persistent. [17]

4.1.7 Prístup k objektom v databázy

Táto kapitola čerpá s literatúry [14]

V kapitole zaoberajúcej sa transakciami JDO sme si ukázali spôsob vytvorenia nového objektu. Pomocou volania metódy `makePersistent` sme ho dokázali uložiť do databázy. V tejto časti si ukážeme ako môžeme uložené objekty z databáze znovu načítať. Operácia prístupu k objektom v databázy prebieha opäť v rámci transakcie. JDO poskytuje tri možnosti ako daný objekt z databáze získať:

1. *prístup cez identifikátor objektu*
2. *prístup cez extent*
3. *dotaz nad databázou*

4.1.7.1 Prístup cez identifikátor objektu

Tento prístup používame v tom prípade, pokiaľ máme OID objektu. Identifikátory objektov však často generujeme sami, takže sme schopní si vytvoriť OID daného objektu. Objekt získame volaním metódy `getObjectById()`. Má dva parametre. Prvý je OID požadovaného objektu. Druhý parameter je typu boolean. Pokiaľ je true je objekt vždy načítaný priamo z databázy, inak sa najprv kontroluje či náhodou objekt nie je vo vyrovnávajúcej pamäti.

4.1.7.2 Prístup cez extent

Extent triedy je množina všetkých inštancií danej triedy. Extent danej triedy získame volaním metódy `getExtent()`. Metóda má dva parametre. Prvý je objekt typu Class, reprezentujúci triedu, z ktorej požadujeme extent. Druhý parameter je typu boolean. Tento parameter určuje či sa načítajú aj objekty odvodených tried alebo nie.

4.1.7.3 Dotaz nad databázou

JDO používa k dotazovaniu JDO Query Language (JDOQL), ktorý umožňuje špecifikovať množinu objektov uložených v databázy na základe rôznych kritérií. S dotazmi v jazyku JDOQL pracujeme pomocou triedy jdo.*Query*. Volaním metódy `newQuery()` objektu *PersistenceManager* získame objekt triedy Query. Táto metóda má 2 parametre. Prvý je trieda objektov, nad ktorou daný dotaz budeme vykonávať. Druhý parameter je filter, ktorý špecifikuje obmedzujúcu podmienku vzťahujúcu sa k určitej vlastnosti danej triedy objektov.

Pre vykonanie dotazu stačí zavolať metódu `execute()` nad získaným objektom triedy Query. Výsledok je kolekcia objektov, splňujúcich filtrovaciu podmienku.

4.1.8 Dátové typy JDO

Pri ukladaní objektov do databázy je potreba vedieť, aké dátové typy u atribútov perzistentných tried sú podporované v JDO. JDO podporuje nasledujúce dátové typy: [15]

- Primitívne dátové typy: *boolean, byte, short, int, long, char, float, double*
- Základné dátové typy: *Boolean, Character, Byte, Short, Integer, Long, Float, Double, String, Locale, BigDecimal, BigInteger.*
- Kolekcie: *Collection, Set, HashSet*

4.1.9 Vzťahy medzi objektmi

JDO umožňuje modelovanie vzťahov *1:1, 1:N a M:N.* Predstavíme si len vzťah 1:N, ostatné vzťahy sú analogické.

V rámci vzťahu 1:N ponúka JDO tri možností mapovania tohoto vzťahu:

- *Normálne mapovanie* – trieda A obsahuje ako atribút kolekciu objektov triedy B
- *Jednostranné inverzné mapovanie* – každý objekt triedy B obsahuje ako atribút objekt triedy A s ktorým je vo vzťahu.
- *Obojstranné inverzné mapovanie* – jedná sa len o kombináciu predchádzajúcich možností

Podrobnejší popis vzťahov v JDO nájdete v literatúre [16]

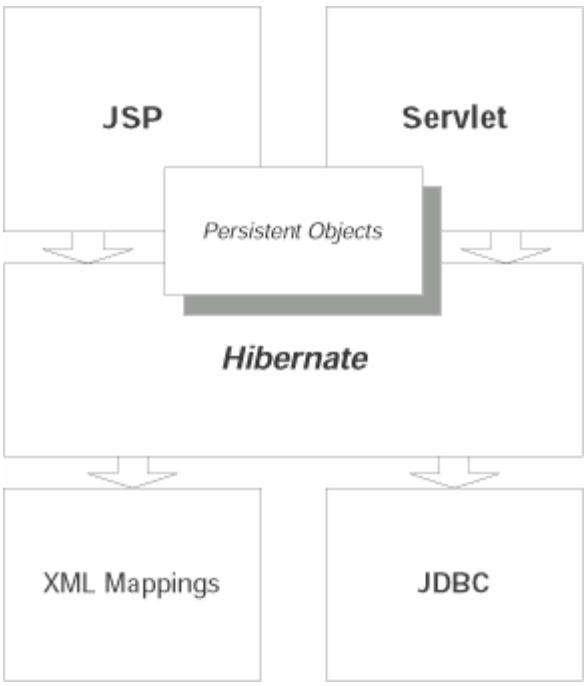
4.2 Hibernate

Hibernate je najrozšírenejší Open Source ORM framework, ktorý poskytuje objektovo relačné mapovanie pre Javovské prostredie. Okrem mapovania dát, reprezentovaných java triedami, na relačné štruktúry, reprezentované databázovými tabuľkami, umožňuje Hibernate získavanie databázových dát na základe vlastného, plne orientovaného jazyka. Tento jazyk sa nazýva Hibernate Query Language (HQL) a je syntakticky veľmi podobný SQL. Je to určený dotazovaniu sa nad databázu, respektíve k vyhľadávaniu a filtrovaniu uložených dát.

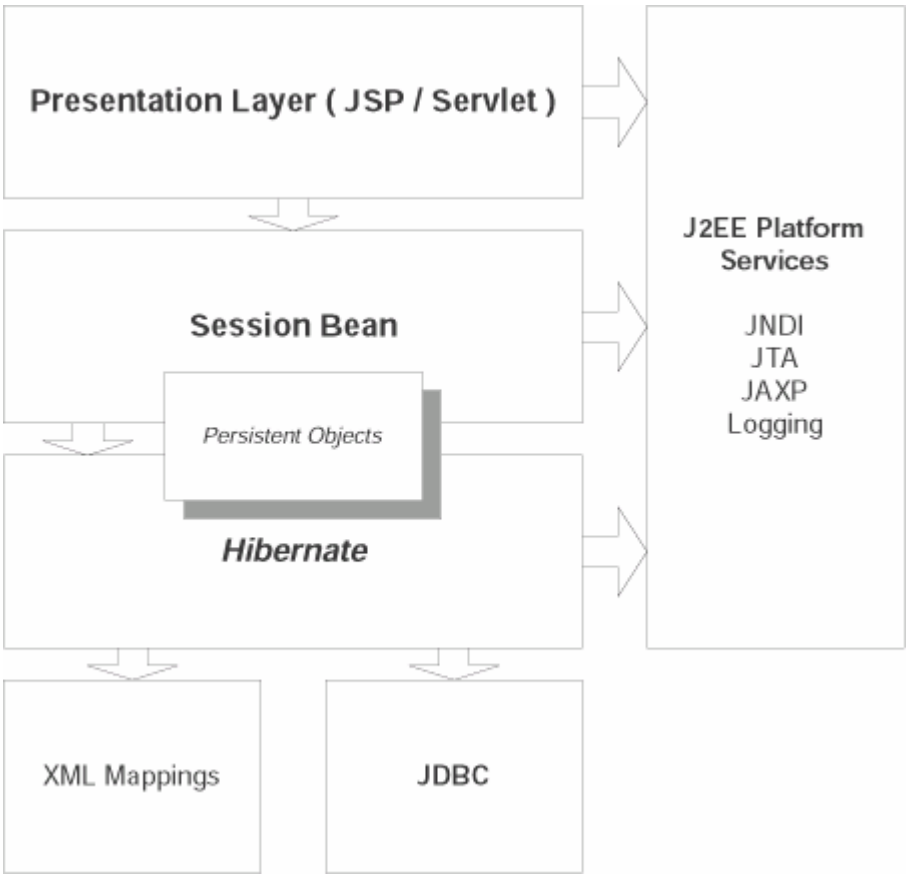
Hibernate predstavuje jednoduché rozhranie, ktoré umožňuje mapovanie do objektov do množstva podporovaných relačných databáz: Oracle, DB2, MySQL, PostgreSQL, Sybase, Microsoft SQL Server, Informix, Ingres atď... . [8]

Hibernate umožňuje ukladať rôzne druhy objektových asociácií, opozdenie inicializácie (nahrá dáta do objektu až keď je to potreba – podobné stavu Hollow u JDO inštancií), umožňuje prácu s kolekciami. Taktiež umožňuje používať užívateľom definované typy, rôzne druhy identifikátorov. Umožňuje mapovanie vzťahov (*one-to-one, many-to-one*). Je možné si nechať na základe mapovacích súborov vygenerovať schéma databáze. [7]

Hibernate je možné použiť v mnohých aplikačných architektúrach. Môžeme ho použiť jednak v klasických desktopových aplikáciach, tak aj v serverových aplikáciach založených a J2EE. Odporúča sa hibernate používať ako persistentnú vrstvu v dvojvrstvovej webovej architektúre (*obrázok č. 6*) a v trojvrstvovej enterprise architektúre (*obrázok č. 7*).



Obrázok č. 6 – webová architektúra [8]



Obrázok č. 7 – Enterprise architektúra [8]

4.2.1 Prístupy k vývoju aplikácie

Hibernate poskytuje 4 základné prístupy k vývoju aplikácií: *Top-down, Bottom-up, Middle-out, Meet-in-the-middle.*

1. **Top-down –** Najprv implementujeme javovský objektový model. Následne môžeme ručne vytvoriť XML mapovací dokument, alebo ho vygenerovať pomocou `Xdclet` tagov. Potom môžeme pomocou nástroja `hbm2ddl` vygenerovať databázové schéma a exportovať ho do konkrétnej databáze.
2. **Bottom-up** – Pokiaľ už máme navrhnuté databázové schéma, môžeme použiť nástroj `Middlegen`, pre vygenerovanie mapovacieho dokumentu. Pomocou nástroja `nhm2java` vygenerujeme kostru javovského kódu, do ktorého už len doplníme vlastnú business logiku danej aplikácie.
3. **Middle-out** – Začíname napísaním mapovacieho dokumentu. Následne z neho pomocou už spomínaných nástrojov `hbm2ddl` a `nhm2java` vygenerujeme kostru java kódu a databázové schéma.
4. **Meet-in-the-middle** – Jedná sa o najťažší prístup. Je navrhnutý objektový model, aj databázové schéma. Je potrebné napísať ručne mapovací dokument, alebo ho vygenerovať, ale je veľmi pravdepodobné, že bude potreba meniť oba modely.

4.2.2 Príklad konfigurácie a mapovania v Hibernate

Na základe toho, že hibernate bol navrhnutý pre prácu v množstve odlišných prostredí, existuje pre neho veľké množstvo konfiguračných parametrov. Hibernate preto obsahuje vzorové súbory, ktorých sú ukážky rôznych nastavení.

Príklad nastavenia konfiguračného súboru Hibernate (*hibernate.cfg.xml*) pre prácu s MySQL pomocou JDBC spojenia:

```
<hibernate-configuration>
 <session-factory>
    <property name="connection.driver_class">
          com.mysql.jdbc.Driver
    </property>
    <property name="connection.url">
          jdbc:mysql://jmeno_serveru:3306/jmeno_databze
    </property>
    <property name="dialect">
          net.sf.hibernate.dialect.MySQLDialect
    </property>
    <property name="connection.username">uzivatel</property>
    <property name="connection.password">heslo</property>
    <mapping resource="Person.hbm.xml"/>
 </session-factory>
</hibernate-configuration>
```

Javovské triedy popísáné v mapovacom súbore (znamená to, že budú mapované do tabuliek relačnej databázy) sú jednoduché JavaBeany zvané POJO (*Plain Old Java Object*). Vzhľadom k tomu že vlastnosti týchto tried sú všetky private, pre každú vlastnosť je potreba implementovať get a set metódy, ktoré nastavia resp. vrátia hodnotu daného atribútu.

Príklad triedy ktorej objekty chceme mať perzistentné:

```
package org.people;
public class Person {
     private integer id;
     private String name;
     private integer age;

     public Person() {}
     public integer getId() {   return id;    }
     private void setId(Integer id) {  this.id = id;    }
```

```
      public String getName() {  return name; }
      public void setName(String name) {  this.name = name; }
      public integer getAge() {  return age;  }
      public void setAge(integer age) {  this.age = age;  }
}  //class Person
```

Mapovanie javovských tried na tabuľky v DB sa prevádza pomocou XML súboru umiestneného adresári s mapovacími triedami. Tieto súbory sú zaznamenané v konfiguračnom súbore *hibernate.cfg.xml* pomocou elementu *<mapping resource="trieda.hbm.xml" />*. Mapovacie súbory je možné písať ručne, alebo použiť niektorý z nástrojov, ktoré ich generujú automaticky. Mená týchto súborov musia odpovedať hodnote elementu mapping v konfiguračnom súbore. Mapovací súbor pre náš prípad bude mať názov *Person.hbm.xml* a obsah:

```
<hibernate-mapping>
<class name="org.people.Person" table="Person">
      <id name="id" type="sequence" unsaved-value="null" >
            <column name="id" sql-type="int" not-null="true"/>
      </id>
      <property name="name">
            <column name="name" length="30" not-null="true"/>
      </property>
      <property name="age"/>
</class>
</hibernate-mapping>
```

V každej namapovanej triede musí byť deklarovaný primárny kľúč. Každá vlastnosť triedy musí byť zachytená elementom *<property>*, ktorý ju mapuje na príslušný atribút databázovej tabuľky. Atribút type potom určuje typ tejto vlastnosti. Hibernate podporuje všetky základné javové dátové typy, ale hodnotou tohoto atribútu môže byť aj javovský objekt. Pokiaľ typ nie je zadaný, hibernate sa ho pokúsi určiť sám. Súčasťou mapovania tried je často aj zachytenie ich vzájomných vzťahov (*1:1, 1:N*) a ďalších vlastností.

Aby sme hibernate mohli použiť v našej aplikácií, musíme si vytvoriť inštanciu triedy *SessionFactory*, ktorá sa nastaví na základe vlastností v konfiguračnom súbore hibernate. Potom vytvoríme *Session* ako pracovnú jednotku hibernate, spustíme *transakciu*. Nasleduje určitý dotaz nad databázu a potvrdenie transakcie a uzavretie *Session*. Zvyčajne sa transakcia uzatvára do chráneného bloku `try {} catch ((HibernateException e) {},` z toho dôvodu, že pokiaľ nastane výnimka, tak sa nad transakciou volá operácia `rollback()`.

```
try {
     session = sessionFactory.openSession();
     tx = session.beginTransaction();
     session.save( new Student("Vit Hamtil", 1981));
     tx.commit();
} catch (HibernateException e) {
     if (tx!=null) tx.rollback();
} finally {
     session.close();
}
```

Hibernate je najrozšírenejší rámec pre objektovo-relačné mapovanie. Jeho použitie je v celku jednoduché. Hibernate je pre Java aplikácie prakticky ORM štandard. Je to nástroj, ktorý nám pri správnom použití môže výrazne zjednodušiť prácu s dátami na databázou.

5 Návrh objektovo – relačného rámca pre PHP

Ťažisko tejto práce spočíva vo využití znalostí už prezentovaných objektovo – relačných rámcov, k návrhu nového ORR pre PHP.

5.1 Analýza ORR

Princíp ORR je práca v objektovom prostredí a ako skladisko dát pritom využívať relačné databázy. Nami navrhovaný ORR bude knižnica PHP (PHP súbor), ktorý inicializuje pripojenie k databázy a musí zabezpečiť vytvorenie databázy. Taktiež musí zabezpečiť, aby následne v aplikácií bolo možné ihneď vytvárať objekty persistentného typu.

Pre pripojenie k databázy sa budú využívať dáta špecifikované v konfiguračnom súbory. Samotné pripojenie bude realizované pomocou PEAR::DB, vďaka čomu dokážeme zabezpečiť jednoduchšiu portabilitu rámca, pri prípadnom prechode na iný druh databázového systému.

Pre objekty, ktoré budeme ukladať do databázy bude potreba špecifikovať súbor, ktorý ich bude popisovať. Tento súbor bude obsahovať všetky vlastnosti, ktoré u objektov požadujeme. V tomto súbore taktiež budeme definovať prípadné vzťahy medzi objektmi. Prakticky sa jedná o súbor ktorý detailne a presne popíše schéma databáze, do ktorej budeme objekty mapovať. Inými slovami sa jedná o konfiguračný súbor metadát.

Oba súbory, či už konfiguračný súbor pre databázu, alebo metadáta persistentných objektov sa budú nachádzať v definovanom adresári.

Knižnica ORR, sa bude do aplikácie vkladať buď v indexe, alebo v inom moduly aplikácie, ktorý je volaný vždy, keď sa bude pracovať s pesistentnými objektmi. Následne sa vytvorí inštancia reprezentujúca rámec, ktorej metódy budú inicializovať objektovo – relačný mapper. Inicializácia rozparsuje definované konfiguračné súbory a inicializuje spojenie na DB a taktiež definované triedy v metadátach, aby následne bolo v aplikácií možné vytvárať priamo objekty popisovaných tried a dané objekty mali všetky svoje popísané vlastnosti.

ORM musí umožňovať vytárať nový objekt danej triedy, tento objekt bude mať všetky vlastnosti vrátene svojho identifikátora prázdne. Po uložení bude do objektu pridaný identifikátor (*ID*), ktorý bude predstavovať primárny kľúč do tabuľky reprezentujúcej triedu daných objektov.

ORM musí taktiež umožniť načítanie objektu z databázy, a to buď na základe identifikátora objektu, alebo načíta zoznam objektov danej triedy. Pri získavaní objektov z databázy bude možné definovať parametre, ktoré sa budú vzťahovať k daným vlastnostiam objektov, a z databázy sa následne načítajú len objekty splňujúce dané podmienky.

V ORM bude možné definovať vzťahy medzi objektmi. Tieto vzťahy budú špecifikované v konfiguračnom súbory metadát pre popis tried objektov. Systém bude podporovať vzťahy typu 1:1 a 1:N. Podporované vzťahy budú môcť byť inverzné a neinverzné, jednostranné aj obojstranné. Vzťahy budú modelované pomocou cudzích kľúčov odkazujúcich sa na primárne kľúče tabuliek (tried), s ktorými sú vo vzťahu. Pri ukladaní objektu, ktorý má položku obsahujúcu vzťahu odpovedajúci objekt, je spolu s ukladaným objektom uložený aj objekt, ktorý je s ním vo vzťahu.

5.2 Konfiguračné súbory ORR

Konfiguračný súbor, ktorý bude špecifikovať vlastnosti potrebné k pripojeniu na databázu a taktiež súbor obsahujúci metadáta pre schéma databázy bude vo formáte XML. Tento typ súboru je vhodný vzhľadom k tomu, že je ho možné jednoducho parsovať a taktiež sa jednoducho vytvára, respektíve upravuje.

5.2.1 Konfiguračné XML pre databázový server

Jedná sa o konfiguračný XML súbor, ktorý obsahuje všetky parametre potrebné k pripojeniu sa na databázový server pomocou triedy PEAR::DB. Cesta k súboru je definovaná pri inicializácií ORR. Pre pripojenie k MySQL bežiaceho na localhoste bude mať konfiguračný súbor nasledujúci obsah:

```
<?xml version="1.0" encoding="UTF-8"?>
<dbconfig name=”local_mysql” server=”master”>
     <dbtype> mysql </dbtype>
     <host> localhost </host>
     <database> inzerce </database>
     <user> login </user>
     <password> password </password>
</dbconfig>
```

- <***dbtype***> - nám určuje typ daného databázového servera, na ktorý sa budeme pripájať. V tomto prípade sa jedná o MySQL server.
- <***host***> - určuje host na ktorom beží daný databázový server. V našom prípade sa jedná o localhost.
- <***database***> - určuje databázu na ktorú sa budeme pripájať.
- <***user***> - užívateľské meno pod ktorým sa budeme prihlasovať k databázy.
- <***password***> - heslo pod ktorým sa budeme prihlasovať k databázy.

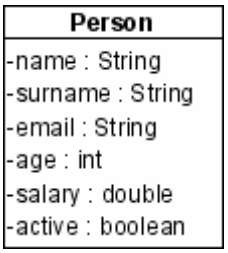
Všimnime si, že v konfiguračnom súbore už zadávame názov databázy, ku ktorej sa budeme pripájať. Databázu musíme vytvoriť ručne. Samozrejme môže byť úplne prázdna.

Samozrejme iné typy DB môžu vyžadovať aj iné (podrobnejšie) konfiguračné nastavenia. Vždy stačí zadať štandartné údaje, a ostatné nastavenia budú automaticky implicitné. Pre bližšie informácie odporúčame manuál PEAR a konkrétne špecifikáciu triedy DB.

5.2.2 Metadáta pre popis tried

Triedy, ktorých objekty budú persistentné, je potreba špecifikovať. Táto špecifikácia je uložená v XML súbore metadát pre popis tried. Cesta k súboru je definovaná pri inicializácií ORR. Spôsob akým sa jednotlivé triedy popisujú v XML metadátach si ukážeme na nasledujúcom príklade.

Predstavme si triedu Osoba (*obrázok č. 8*), ktorá má atribúty: *name, surname, email, age, salary* a boolean atribút *active* identifikujúci či je daná osoba stále aktívna.



Obrázok č. 8 – trieda Osoba

Metadáta pre túto triedu budú mať nasledujúcu formu:

```
<?xml version="1.0" encoding="UTF-8"?>
<package name="inzerce">
<class name="Person" create="create-drop" engine="MyISAM">
     <field name="name" type="string" length="30"></field>
     <field name="surname" type="string" length="30"></field>
     <field name="email" type="string" length="50"></field>
     <field name="age" type="smallint" null="null"></field>
     <field name="salary" type="double" length="10,2"></field>
     <field name="active" type="boolean" default="1"></field>
</class>
</package>
```

Z uvedeného príkladu je zrejmé, že jednotlivé triedy charakterizuje element `<class>`, ktorý obsahuje atribúty:

- *name* – názov triedy a tabuľky v databázy.
- *engine* – špecifikuje typ databázovej tabuľky, môže nadobúdať všetky hodnoty, ktoré daný databázový systém poskytuje,
- *create* – charakterizuje čo sa s tabuľkou deje pri opätovnom volaní. Tento atribút bude vo väčšine prípadov úplne vynechaný. Pretože by sa v aplikácií prakticky neudržali

dáta, avšak môže byť vhodný napríklad pri testovaní. Môže nadobúdať hodnoty: *create-drop* - odstráni a vzápätí vytvorí danú tabuľku, *drop* - odstráni tabuľku z databázy, *create* – ponechá tabuľku nezmenenú aj v prípade zmeny metadát (implicitná hodnota ktorá sa nemusí zadávať)

Jednotlivé vlastnosti triedy sú implementované pomocou elementu `<field>`, ktorý obsahuje nasledujúce atribúty:

- *name* – povinný atribút, špecifikuje názov vlastnosti objektu a názov domény v danej tabuľke.
- *type* – špecifikuje dátový typ danej vlastnosti. Tento atribút môže nadobúdať nasledujúce hodnoty: *string, itenger, double, boolean, object* a *collection*. Z hľadiska vlastností objektu je jasné, že každá z možností predstavuje daný dátový typ, vyplývajúci z jej názvu. V databázy sa však ako typy domén môžu mapovať rôzne. *String* sa do databázy mapuje tak, že u danej domény nastaví typ *text*, alebo *varchar* (pokiaľ je definovaný aj atribút length). *Boolean* sa namapuje ako typ *tinyint* s dĺžkou jedna, kde hodnota 1 bude predstavovať *true* a hodnota 0 *false*. *Object* a *collection* sú dôležité pri modelovaní vzťahov. Vzťahy podrobne popíšeme v nasledujúcej časti.
- *length* – stanuje prípadnú maximálnu dĺžku u dátových typov, u ktorých je to možné nastaviť. Tento atribút má vplyv len na databázu. Kombinácia typu String a určitej dĺžky sa do databázy premietne ako typ *varchar(dlžka)*.
- *default* – stanovuje default hodnotu, na ktorú sa nastaví vlastnosť objektu v prípade vytvárania nového objektu.
- *null* – implicitne je nastavený na NOT NULL, takže ho nastavujeme len ak chceme explicitne definovať, že daný atribút môže v databázy nadobúdať nulovej hodnoty.

Samotný element `<class>` je zaobalený do elementu `<package name="inzerce">`, ktorého atribút *name* špecifikuje databázu v ktorej sa tabuľka nachádza.

Pokiaľ systém nenájde v atribúte *type* ani jednu z uvedených podporovaných hodnôt tak pri vytváraní databázy použije do SQL príkazu ako typ danej domény priamo hodnotu tohoto atribútu. Tým pádom môžeme vytvárať všetky typy domén, ktoré podporuje daná databázový model. Tento postup sa však nedoporučuje, vzhľadom k tomu, že pri podporovaných hodnotách atribútu type sa vlastnosti daného objektu nastavia typovo správne, avšak pri použití inej, než podporovanej hodnoty bude vlastnosť objektu bez dané typu. Táto vlastnosť nás však nemusí veľmi trápiť, vzhľadom k nie veľmi dokonalej typovej kontrole PHP.

5.3 Vzťahy medzi objektmi v ORR

Základom návrhu dnešných aplikácia je z hľadiska aplikácií diagram tried, tento diagram graficky reprezentuje jednotlivé triedy aplikácie a ich vzťahy. Tieto vzťahy je nutné modelovať taktiež v databázy. Ako sme však už upozorňovali v kapitole č. 2, u relačných databáz je relatívne zložité vytváranie vzťahov. Je potreba spájať tabuľky na základe rovnakých hodnôt určitých stĺpcov v rámci relačných databáz. Tomuto sa pri mapovaní samozrejme nevyhneme, avšak hlavnú časť práce podsunieme pod O-R rámec. Pri tvorbe metadát stačí definovať vzťah k inej triede a všetko ostatné za nás už spraví ORR.

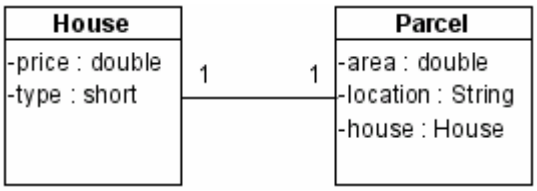
ORR podporuje nasledujúce druhy vzťahov:

- Vzťah 1:1 – ku každému objektu triedy A existuje práve jeden objekt triedy B
- Vzťah 1:N – ku každému objektu triedy A môže byť priradených viacej objektov triedy B, vrátane žiadneho.
- Vzťah N:M – zatiaľ nie je podporovaný. Bude navrhnutý v rámci rozšírenia.

5.3.1 Vzťahy 1:1

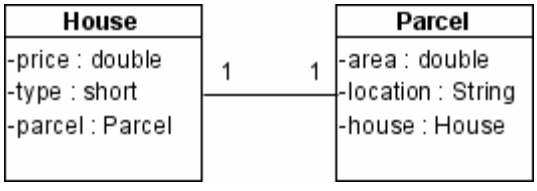
Tieto vzťahy môžu byť dvojakého druhu: *Jednostranný vzťah* a *Obojstranný vzťah*. V prípade jednostranného vzťahu obsahuje objekt triedy A informáciu o odpovedajúcom objekte triedy B, ale nie opačne. Pokiaľ modelujeme obojstranný vzťah, tak objekt trieda A obsahuje referenciu na objekt triedy B, ale zároveň aj objekt triedy B obsahuje referenciu na objekt triedy A.

Uvažujme príklad vzťahu 1:1 dom a pozemok (triedy House a Parcel), vynecháme možnosť, že na jednom pozemku by mohlo stáť viacej domov. V prípade modelovania jednostranného vzťahu by bola informácia o dome ktorý leží na pozemku len v triede Parcel (*obrázok č. 9*).



Obrázok č. 9 – znázornenie jednostranného vzťahu 1:1 medzi dvoma triedami

V prípade modelovania obojstranného vzťahu, by v triede House pribudla položka odkazujúca sa na triedu Parcel (*obrázok č. 10*).



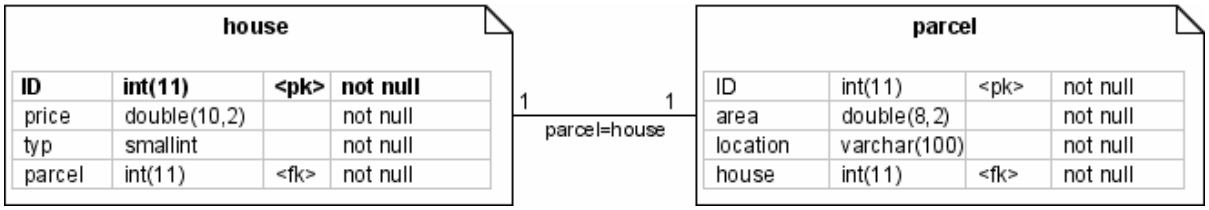
Obrázok č. 10 – znázornenie obojstranného vzťahu 1:1 medzi dvoma triedami

Nasledujúci kód demonštruje popis spomínaného vzťahu v metadátach nášho ORR. V metadátach je znázornený obojstranný vzťah.

```
<class name="House" engine="MyISAM">
     <field name="price" type="double" length="10,2"></field>
     <field name="typ" type="smallint"></field>
     <field name="parcel" type="object" class="Parcel"></field>
</class>

<class name="Parcel" engine="MyISAM">
     <field name="area" type="double" length="8,2"></field>
     <field name="location" type="string" length="100"></field>
     <field name="house" type="object" class="House"></field>
</class>
```

Všimnime si elementy `<field>`, ktoré majú v atribúte *type* hodnotu *object*. Pokiaľ je u tohoto atribútu uvedená práve hodnota object, ORR očakáva ešte definíciu triedy v atribúte *class* ku ktorej sa daná trieda viaže. Z uvedeného kódu je zrejmé, že jednostranný vzťah by sa modeloval obdobne, napríklad bez odkazujúcej sa položky *parcel* v triede House. ORR sám pripraví do tabuliek relačnej databázy odpovedajúce domény, odkazujúce sa definované triedy. Nasledujúci obrázok znázorňuje ORR automaticky vytvorené databázové schéma na základe vyššie uvedených metadát



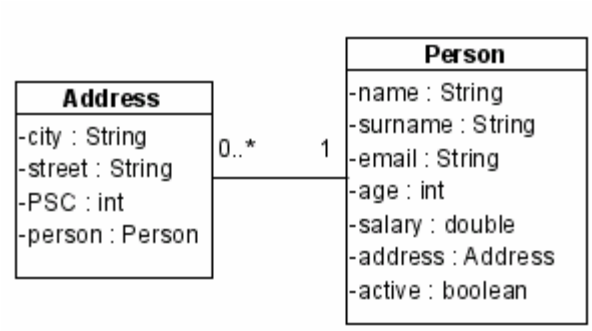
Obrázok č. 11 – Logické schéma databáze pre vzťah tried znázornených na obr.č.10

5.3.2 Vzťahy 1:N

Pri tomto druhu vzťahu, je jednému objektu triedy A priradených viac objektov triedy B, prípadne žiadny objekt. ORR je schopné zabezpečiť len jeden typ mapovania tohoto vzťahu a to *Obojstranné inverzné mapovanie* - trieda A obsahuje atribút reprezentujúci kolekciu objektov triedy B a zároveň každý objekt triedy B obsahuje referenciu na k nemu sa vzťahujúci objekt triedy A.

V rámci ORR je kolekcia predstavovaná iterátorom PHP objektu ArrayObject. V aplikačnej business logike nám podpora len jedného typu tohoto vzťahu plne postačí, pretože Jednostranné inverzné mapovanie, resp. normálne mapovanie sú „silovo slabšie“ spôsoby implementácie vzťahu 1:N.

Opäť uvažujme príklad tentoraz vzťahu 1:N. Majme triedu Person a triedu Address. Berieme do úvahy, že osoba môže mať viac adries.



Obrázok č.12 – Diagram tried - znázornenie obojstranného inverzného vzťahu 1:N

Metadáta popisujúce tento vzťah budú mať nasledovnú formu:

```
<class name="Person" create="create-drop" engine="MyISAM">
      <field name="name" type="string" length="30"></field>
      <field name="surname" type="string" length="30"></field>
      <field name="email" type="string" length="50"></field>
      <field name="age" type="integer"></field>
      <field name="salary" type="double" length="10,2"></field>
      <field name="address" type="collection" class="Address"></field>
      <field name="active" type="boolean" default="1"></field>
</class>

<class name="Address" create="create-drop" engine="MyISAM">
      <field name="city" type="string" length="50"></field>
      <field name="street" type="string" length="50"></field>
      <field name="PSC" type="string" length="5"></field>
      <field name="person" type="object" class="Person"></field>
</class>
```

Môžeme si povšimnúť, že pri atribúte, ktorý bude reprezentovať kolekciu objektov je uvedený typ collection. Keď ORR narazí na typ collection u atribútu objektu inicializuje v danom atribúte ArrayObject. Ku kolekciám objektov sa pristupuje metódou getNazovAtributu(), ktorá následne vracia iterátaror poľa objektov, alebo false v prípade, že je pole prázdne, alebo nebolo inicializované

5.3.3 Vzťahy N:M

Tento vzťah síce nie je v ORR podporovaný, ale pre bežné aplikácie by mal vystačiť vzťah 1:N. V prípadnom rozšírení ORR by samozrejme tento vzťah už bol podporovaný.

5.4 Princíp ORR

Samotný objektovo-relačný rámec pozostáva len z jednej knižnice (PHP modulu). V tejto knižnici sú implementované tri základné triedy. Tieto triedy zabezpečujú funkčnosť celého ORR. Jedná sa o triedy *dbase*, *ORR* a *ORM*. Podrobnejšie tieto triedy popisujeme v kapitole č. 6.

5.4.1 Spôsob použitia ORR

Ako sme už spomenuli ORR pozostáva z jednej knižnice. Keďže však je potreba spustiť skupinu metód, tak je výhodné knižnicu volať v určitom inicializačnom súbore, v ktorom sa spustia dané konfiguračné metódy. Do samotnej aplikácie potom už len budeme vkladať tento inicializačný súbor.

Príklad takéhoto súboru – nazveme ho *init.php* :

```
require_once ("orr.php");
$orr=new ORR();

$orr->setDB_config_file("db_xml/db_config.xml");
$orr->setDB_metadata_file("db_xml/db_metadata.xml");
$db=$orr->set_db_connection(true);
$orr->db_touch=true;
$orr->private_vars=false;
$orr->generate();
```

Po vytvorení objektu ORR triedy, nastavujeme cesty ku konfiguračným súborom. Následne voláme metódu `set_db_connection()`, ktorá nastaví pripojenie k databázy a uloží inštanciu triedy PEAR::DB, ktorá sa stará o spojenia s databázou, do privátnej premennej `$dbConn` triedy dbase a taktiež do statickej premennej triedy ORR. Pomocou tejto statickej premennej budeme pristupovať k databázy v rámci ORM triedy a v rámci staticky volaných metód triedy ORR. Metóda má jeden nepovinný parameter typu boolean. Pokiaľ je true, tak metóda vracia daný objekt triedy dbase. Pomocou získaného objektu môžeme následne pristupovať k databázy mimo ORR. Prioritou však je pristupovať len pomocou metód v ORR.

Vlastnosť `$db_touch` je dátového typu boolean a určuje, či pri opätovnom volaní aplikácie dôjde znovu k vytváraniu databáze (analogické parametru create v špeifikácií metadát). Odporúča sa po prvom spustení aplikácie premennú nastaviť na false.

Vlastnosť `$private_vars` je opäť typu boolean, a umožňuje nastaviť či vlastnosti perzistentných tried špecifikované v metadátach budú private alebo nie. Implicitne je nastavená na true.

Metóda `generate()` predstavuje spúšťaciu metódu ORR, po volaní tejto metódy, dochádza ku generovaniu databáze, pokiaľ nie je vypnutá vlastnosť `$db_touch`. Na rozdiel od generovania databáze dochádza vždy ku príprave perzistentných tried. Ich kód je vykonaný. Pri prechádzaní metadát sa taktiež do ORR nadefinovali vzťahy v nich popísané.

V aplikácií nám už len stačí pripojiť daný súbor init.php a môžeme priamo vytvárať objekty z tried popísaných v metadátach. Tieto objekty obsahujú metódy pre prístup ku všetkým svojim vlastnostiam. Vždy sa jedná o metody typu `get/setNázov_vlastnosti()`. Jedinú výnimku medzi vlastnosťami tvoria vlastnosti typu boolean, ktoré namiesto metódy `getVlastnost()` využívajú metódy `isVlastnost()`.

5.4.1.1 Vytváranie objektov v ORR

Objekty sú vytvárane klasickým spôsobom, pomocou operátora `new NázovTriedy()`.

Príklad vytvárania a ukladania objektov pomocou ORR. Príklad popisuje ukladanie objektov triedy Person a Address a ich 1:N vzťahu. (viď obrázok č. 12.)

```
$pers=new Person();
$pers->setName("Michal");
$pers->setSurname("Hudec");
$pers->setAge(23);

$adresa=new Address();
$adresa->setCity("Brno");
$adresa->setPerson($pers);

$adresa2=new Address();
$adresa2->setCity("Praha");
$adresa2->setPerson($pers);

$pers->Save();
```

Vidíme, že vytváranie nových objektov je úplne triviálne. Musíme brať do úvahy fakt, že dané triedy neboli nikde programovo špecifikované, ale ich kód bol automaticky vykonaný v rámci ORR, pomocou definície uvedenej v metadátach tried.

Všimnime si, že boli vytvorené dva objekty triedy Address, obom bol priradený objekt triedy Person. To znamená, že tento objekt obsahuje v atribúte address kolekciu daných dvoch objektov triedy Address. Pri uložení tohto objektu dôjde automaticky k uloženiu objektov ktoré sú s nim vo vzťahu a ešte nie sú perzistentné. Tým pádom je môžme priraďovať do kolekcie množstvo objektov a uložiť ich všetky do databáze pomocou jednej metódy.

5.4.1.2 Prístup k objektom v databázy

ORR poskytuje tieto možnosti prístupu k objektom v databázy:

- Získanie objektu pomocou identifikátora
- Prístup pomocou kolekcie objektov
- Natívne SQL príkazy nad databázou

Získanie objektu pomocou identifikátora

Tento prístup k objektom v databázy používame v prípade, že poznáme identifikátor objektu, ktorý potrebujem načítať z databáze. Trieda ORM poskytuje metódu `getObject()`.Všetky triedy, ktorých objekty majú byť perzistentné sú odvodené od triedy ORM. Takže je možné túto metódu volať z triedy ktorej objekt požadujeme.

```
$parcel=Parcel::getObject(1,"Parcel");
```

Metóda `getObject()` má dva parametre. Identifikátor objektu a názov triedy. Názov triedy je potreba zadávať, pretože metódu voláme staticky. Preto je z programového hľadiska lepšie volať priamo metódu rámca `OpenID()`.

```
$parcel=ORR::OpenID("Parcel",1);
```

Metóda `OpenID()` má dva parametre. Prvý je názov triedy, ktorej objekt požadujeme. Druhý je ID objektu, ktorý požadujeme.

Posledný spôsob ako získať objekt predstavuje volanie metódy `OpenID()` z objektu. V tomto prípade dôjde k nahradeniu stávajúceho objektu objektom požadovaným.

```
$adresa=new Address();
$adresa->setCity("Brno");
$adresa->Save();
$adresa2=new Address();
$adresa2->OpenID(adresa->getObjectID());
```

V danom príklade vytvárame dva objekty adresa, pričom sa vzápätí rozhodneme pre kópiu objektu adresa. Samozrejme muselo medzičasom dôjsť k uloženiu objektu adresa.

Prístup pomocou kolekcie objektov

K tomuto prístupu sa používa metóda `getExtent()`. Táto metóda má tri parametre, všetky sú nepovinné. Prvý parameter predstavuje klauzulu `Where`, ktorá sa ovplyvňuje výsledok SQL dotazu. Jedná sa o určitý filter, ktorý sa vzťahuje na vlastnosti objektov. Druhý a tretí parameter predstavujú hodnoty SQL príkazu `LIMIT`, takže pomocou nich môžeme špecifikovať koľko záznamov vlastne požadujeme.

```
$it=Person::getExtent("salary>30000");
if(!is_null($it)) {
     while($it->valid()) {
          $ob=$it->current();
          echo $ob->getName();
          $it->next();
     }
}
```

V uvedenom príklade sme si nechali vypísať zoznam osôb s platom nad 30 tis. Metóda `getExtent()` inicializuje objekt ArrayObject nad množinou záznamov vrátených z databázy. Po inicializácií ArrayObject objektu automaticky vracia jeho iterátor.

Natívne SQL príkazy nad databázou

Objektovo-relačný rámec síce poskytuje spôsoby ako dané objekty z databázy získavať. Môže však nastať situácia, kedy nám jeho metódy nebudú postačovať. V tomto prípade je nám veľmi nápomocná možnosť nechať si pri inicializácií spojenia vrátiť referenciu na tento objekt. (viď kap. 5.4.1.). Získaný objekt nám poskytuje niekoľko metód pre prístup k databázy. Detailnejšie sú popísané v kapitole 6.

Predstavme si jednoduchý príklad získania hodnoty domény daného záznamu. Metóda `DBGetOne()` sa používa na vrátenie len jednej domény.

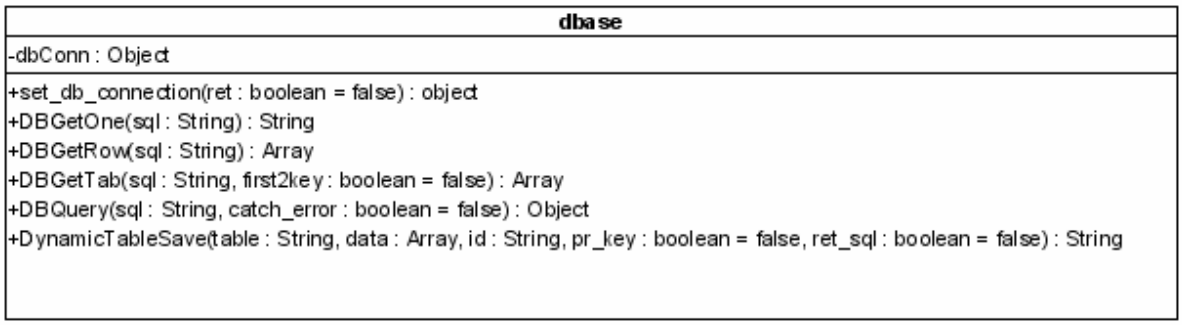
```
$sql="SELECT name FROM Person WHERE ID=1";
echo $db->DBGetOne($sql);
```

6 Implementacia ORR

Ako už bolo spomenuté v návrhu, funkčnosť ORR zabezpečujú tri triedy, dbase, ORM a ORR. V tejto kapitole si popíšeme význam jednotlivých tried. Nebudeme konkrétne popisovať funkčnosť všetkých metód a význam atribútov týchto tried. Budeme sa zaoberať len najdôležitejšími metódami. A popíšeme si princíp akým ORR umožňuje v aplikácií vytvárať inštancie tried popísaných v metadátach.

6.1 Trieda dbase

Zabezpečuje pripojenie k databázy pomocou PEAR::DB. Obsahuje metódy pre prístup k databázy a inštanciu spojenia s databázou.



Obrázok č. 13 – trieda dbase

Táto trieda obsahuje privátnu vlastnosť `dbConn`, ktorá obsahuje inštanciu triedy PEAR::DB, pomocou ktorej ORR komunikuje s databázou. Vytvorenie tejto inšancie zabespečuje metóda `set_db_connection()`. Na získanie parametrov z konfiguračného XML, využíva XML parser triedy ORR. Referenciu na objekt pripojenia k databázy je možné získať volaním metódy `set_db_connection(true)`.

Metódy `DBGetOne()`, `DBGetRow()` a `DBGetTab()` majú za úlohu vracať dáta z dotazu na databázu v jednoduchej podobe, pre ďalšie spracovanie. Tieto funkcie k tomu využívajú aj metódy PEARu pre prácu s databázou. `DBGetOne()`- vracia hodnotu primitívneho dátového typu (viď príklad v predchádzajúcej kapitole). `DBGetRow()`- vracia jeden záznam z relačnej tabuľky vo forme poľa. `DBGetTab()`- vracia množinu záznamov z relačnej tabuľky vo forme poľa.

`DBQuery()` – pošle dotaz na databázový server a vráti výsledok. Využívajú ju všetky metódy popísané vyššie.

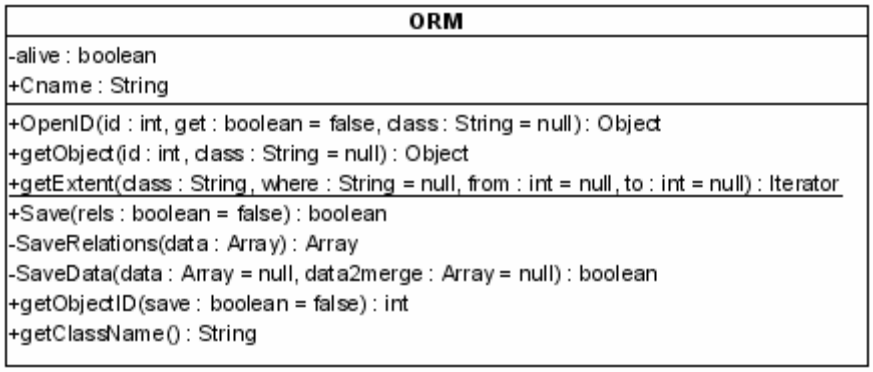
`DynamicTableSave()` – realizuje automatické ukladanie dát do relačných tabuliek. Automatické – je myslené tak, že nepotrebuje priamy dotaz na databázu vo forme INSERT alebo UPDATE, ale postačujú jej tri základné parametre:

- *Názov tabuľky* – parameter typu string (napríklad „Person“)
- *Pole dát* – jedná sa o associatívne pole v tvare („doména tabuľky“=>“hodnota“), pre uloženie záznamu sú relevantné len údaje, ktorých kľúč v danom poli odpovedá jednotlivým doménam, ostatné údaje sú ignorované
- *Názov primárneho kľúča* – zväčša typu string („ID“). V prípade zloženého primárneho kľúča môže byť pole stringov.

Ďalší nepovinné parametre definujú, či sa má pri ukladaní záznamu prepisovať primárny kľuč. Posledný, nepovinný atribút slúži skôr k debugovaniu. Je typu boolean a pokiaľ je true, tak metóda nevykoná výsledný SQL príkaz, ale vráti ho vo forme stringu. Inak vracia ID posledného záznamu pri INSERTE, resp. true ak sa podaril UPDATE záznamu.

6.2 Trieda ORM

Trieda objektovo-relačného mapperu.



Obrázok č. 14 – trieda ORM

Každá trieda ktorej objekty budeme chcieť ukladať do databáze musí byť rozšírená o túto triedu. Obsahuje totiž privátne metódy pre nahrávanie objektov z databázy, či už sa jedná o vracanie samotného objektu alebo kolekcie objektov danej triedy. Obsahuje privátne metódy, ktoré realizujú uloženie objektu do databázy. Obsahuje vlastnosť, ktorá je nositeľom názvu triedy, z ktorej je objekt inicializovaný.

Metóda `OpenID()` – načíta objekt z databáze podľa zadaného ID. Má tri parametre. Prvý je ID objektu, druhý nepovinný parameter je typu boolean, udáva či načítaný objekt prepíše existujúci (v zmysle názvu danej metódy), alebo bude vytvorený nový objekt a vrátená referencia naň. Tretí taktiež nepovinný parameter je názov triedy ktorej objekt chceme získať.

Metóda `GetObject()`- je zaobalením metódy OpenID s nastaveným parametrom pre vracanie referencie na objekt. (v zmysle názvu danej metódy).

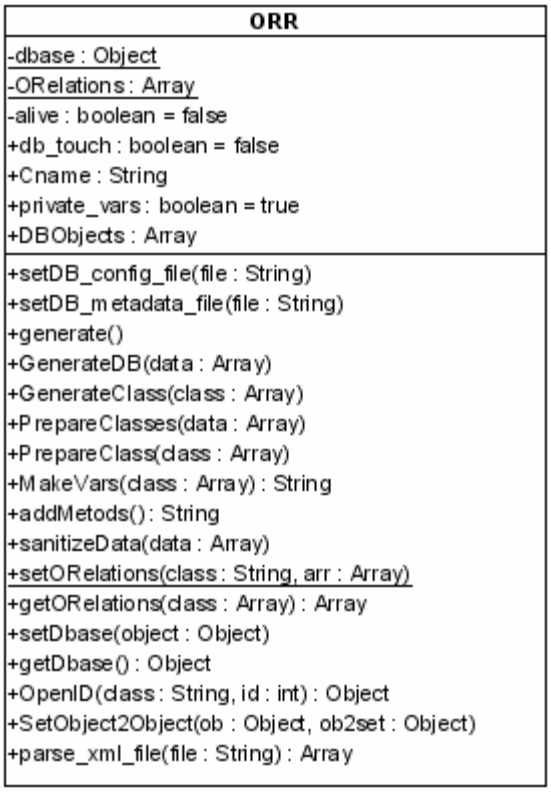
Metóda `getExtent()` – vracia kolekciu objektov, respektíve iterátor do danej kolekcie. Metóda má štyri parametre. Prvý parameter, jediný povinný, je názov triedy, z ktorej danú kolekciu požadujeme. Druhý parameter je filter, ktorý sa vzťahuje na vlastnosti danej triedy. Môže byť typu

String a *Array*. Pokiaľ je typu *Array* tak sa zo zložiek poľa vyskladá výsledný `Where` ktorý sa pridá do SQL. Pokiaľ je typu *String* tak sa automaticky vloží do SQL. Metóda vytvára pole objektov a vracia iterátor do tohoto poľa.

Metóda `Save()` – ukladá objekt do databázy. Volaná priamo z objektu (objekt musí byť inicialzovaný z triedy, ktorá je odvodená od triedy ORM). Môže mať jeden nepovinný parameter typu boolean, ktorý určuje, či sa majú uložiť do databázy aj objekty, ktoré sú s ukladaným objektom vo vzťahu. Implicitne je tento parameter nastavený na true. Pre samotné uloženie objektu do relačnej tabuľky volá privátnu metódu `SaveData()`, ktorá využíva metódu `DynamicTableSave().`

6.3 Trieda ORR

Trieda predstavujúca kostru objektovo-relačného rámca. Táto trieda je rozšírená o triedu dbase. Dôvodom je časté volanie databázy v rámci triedy. Preto je výpočtovo výhodné mať inštanciu triedy zabezpečujúcej operácie nad databázou priamo v lokálnych vlastnostiach. Obsahuje metódy, ktoré nastavujú konfiguračné súbory (viď kap. 5.2), metódy ktoré parsujú dané konfiguračné súbory, metódy zabezpečujúce vznik databázovej schémy, metódy umožňujúce vznik daných perzistentných objektov z tried popísaných v metadátach. Obsahuje privátne vlastnosti, ktoré mapujú vzťahy medzi jednotlivými objektmi. Trieda ORR je zobrazená diagramom tried na obrázku č. 15



Obrázok č. 15 – trieda ORR

Statický atribút `dbase` obsahuje referenciu na objekt komunikujúci s databázou. Tento atribút je naplnení pri volaní metódy `set_db_connection()`.

Statický atribút `ORelations` predstavuje mapovacie pole vzťahov. Musí byť reprezentované statickým atribútom pretože, k nemu pristupujú objekty rôznych tried. K nastavovaniu tohoto atribútu dochádza v metóde `MakeVars()`. Postačuje vedieť, že pre každú triedu obsahuje zoznam tried s ktorými je vo vzťahu. Samozrejme uchováva typ vzťahu a názov vlastnosti cez ktorú je daný vzťah implementovaný.

Atribút `db_touch` už bol spomenutý. Je typu boolean a povoľuje alebo zakazuje prístup k databázy v procese generovania databázy, resp. kódu príslušných perzistentných tried. Inými slovami zabraňuje dodatočnému vytváraniu tabuliek, aj keď boli nanovo pridané do popisu metadát. Nemá žiadnu súvislosť s ukladaním objektov do databázy. Implicitne je nastavený na false, pretože sa predpokladá, že užívateľ si tento atribút zapne len pri prvom volaní aplikácie.

Atribút `private_vars` je typu boolean, a špecifikuje či vlastnosti perzistentných objektov budú mať modifikátor `private`, tým pádom by k nim bol umožnený prístup jedine pomocou get/set metód pre dané vlastnosti.

Po vytvorení inštancie tejto triedy voláme postupne metódy: `setDB_config_file()`, `setDB_metadata_file()`,`set_db_connection()`,`generate()`.

Metódy `setDB_config_file()` a `setDB_metadata_file()` majú ako vstupné parametre cesty ku XML súborom. Samotné metódy XML neparsujú, ale nastavia cesty do konštánt.

Metóda `set_db_connection()` je rozoberaná pri popise triedy `dbase`

Aby sme vôbec ORR mohli v aplikácií používať musíme volať metódu `generate()`. Táto metóda volá `parse_xml_file(DB_METADATA_FILE)`, kde daný parameter je konštanta s cestou k XML súboru metadát, jedná sa interný parser ORR, ktorý využíva XML_Unserializer, knižnice PEAR.

Po získaní dát je volaná `sanitizeData()`, ktorá upravuje určité nedostatky XML parseru. Keď máme dáta ošetrené volá metóda `GenerateDB()`, ktorá pre každú popisovanú tabuľku v metadátach volá `GenerateClass()`. V tejto metóde dochádza k vyskladaniu SQL dotazu pre vznik tabuľky. Databáza sa nemusí generovať, pokiaľ atribút `db_touch` zakazuje.

Následne sa volá metóda `PrepareClasses()`, ktorá pre každú triedu špecifikovanú v XML metadátach volá metódu `PrepareClass()`. Táto metóda postupne generuje kód triedy, ktorej objekty majú byť perzistentné. Tento kód generuje vo forme reťazca. Na konci tejto metódy sa výsledný reťazec kódu perzistentnej triedy vykoná pomocou príkazu `eval()`, čo je v podstate to isté ako keby daná trieda bola napísaná priamo v PHP. Samotné generovanie kódu je rozdelené do troch častí.

- *Prvá časť* - špecifikuje, že daná trieda bude odvodená od triedy `ORM`, čo nám umožňuje v aplikačných objektoch volať metódy triedy ORM (`Save, OpenID, getExtent` a pod.).
- *Druhá časť* – vykonáva ju metóda `MakeVars()`. Ako je zrejmé z názvu metódy, generuje sa v nej kód reprezentujúci jednotlivé atribúty danej triedy a k nim odpovedajúce get/set metódy. (v aplikácií sa potom pristupuje k atribútom nasledovne – getNazov_atributu(), set analogicky). V tejto metóde dochádza k parsovaniu XML kódu špecifikujúceho domény relačných tabuliek, preto tu prebieha ja tzv. mapovanie vzťahov tried. Každý nájdený vzťah sa ukladá do statického atribútu triedy ORR.
- *Tretia časť* – ide o dodatočné metódy, ktoré by sme chceli u perzistentných tried používať. Tento úsek kódu generuje metóda `addMetods()`. Takže prípadné rozširovanie, resp. pridávanie nových metód bude realizované v tejto metóde.

Po skončení metódy `PrepareClass()` skončí následne aj `generate()`. Následné je ORR pripravený na použitie v aplikácií.

Záver

Mali sme za úlohu vytvoriť knižnicu, respektíve Objektovo-relačný rámec pre PHP, ktorý by nám jednoducho a bez znalostí SQL umožnil ukladať objekty do odpovedajúcich relačných tabuliek.

V úvode práce sme popisovali a porovnávali relačné a objektové databázové systémy.

V tretej kapitole sme si uviedli základné požiadavky, ktoré by mal objektovo-relačný rámec spĺnať. Vysvetlili sme si princíp mapovania atribútov objektov do domén relačných tabuliek.

Vo štvrtej kapitole sme sa zamerali na štúdium už existujúcich Objektovo – relačných rámcov. Zamerali sme sa najmä na metadáta popisujúce uloženie objektov v relačnej databázy.

Tieto poznatky sme následne uplatnili pri návrhu nášho objektovo-relačného rámca (ORR), ktorým sa zaoberá piata kapitola. Návrh počíta s metadátami uloženými v XML súboroch. XML sme vybrali, kvôli ich jednoduchému parsovaniu.

Navrhli sme triedy reprezentujúce ORR. Ich metódy sú popísané v šiestej kapitole.

Nami navrhnutý ORR dokáže vytvárať a ukladať objekty tried popísaných v metadátach do databáze. Zároveň nám umožňuje tieto objekty spätne získavať. Ďalej umožňuje vytvárať vzťahy medzi objektmi.

ORR však neposkytuje len podmienky pre prácu s objektmi tried špecifikovaných v metadátach. Vďaka jeho spôsobu pripojenia k databázy a pomocou metód triedy `dbase` je výrazne uľahčená práca s databázou. A to aj v momente keď daný ORR nevyužívame pre prácu s objektmi.

Pripojenie k databázy zabezpečuje PEAR, čím sme výrazne podporili portabilitu ORR. Systém ako taký výrazne zjednodušil prácu s objektmi v PHP.

V ďalšom rozšírení systému, by bolo vhodné doplniť možnosť mapovania vzťahu N:M, ktorý systém rámec zatiaľ nepodporuje.

Zaujímavým prvkom by bola možnosť dedičnosti tried popísaných v metadátach. Dedičnosť by mohla byť implementovaná dvoma spôsobmi. V prvom prípade by sa do odpovedajúcej relačnej tabuľky odvodenej triedy pridali domény z odpovedajúcej tabuľky triedy, ktorá danú triedy rozširuje. V druhom prípade by sa do relačnej tabuľky triedy z ktorej budeme dediť pridal identifikátor, ktorý by mohol odkazovať na odpovedajúci záznam do relačnej tabuľky odvodenej triedy.

Danú problematiku by samozrejme bolo potrebné ešte podrobnejšie analyzovať.

Literatura

[1] Hruška, T. Objektovo orientované databázové systémy 1. diel, Informačné systémy, 2006, s. 7-29.

[2] Zendulka, J. 2 Objektově-relační databáze, Pokročilé databázové systémy, 2005, s. 4-7.

[3] Skřiván, J. http://interval.cz/clanky/databaze-nejsou-jen-mysql/

[4] Lávička, J. http://nb.vse.cz/~zelenyj/it380/eseje/xlavj01/od.htm, leto 1999/2000

[5] http://en.wikipedia.org/wiki/Object_relational_mapper

[6] http://computing-dictionary.thefreedictionary.com/O-R+mapping

[7] Jirsák, F. http://nb.vse.cz/~zelenyj/it380/eseje/xjirf01/xjirf01.htm

[8] Hamtil, V. http://nb.vse.cz/~zelenyj/it380/eseje/xhamv01/xhamv01.htm

[9] Hádek, L. http://nb.vse.cz/~zelenyj/it380/eseje/xhadl05/JDO.htm

[10] Klír, O. http://nb.vse.cz/~zelenyj/it380/eseje/xklio04/jdo.htm

[11] Kinský, F. http://www.dbsvet.cz/view.php?cisloclanku=2004042301

[12] Roos, R. http://www.webel.com.au/jdo/Robin_Roos_all_states.jpg

[13] Burget, R. http://interval.cz/clanky/jdo-ruzne-implementace-a-jejich-pouziti/

[14] Burget, R. http://interval.cz/clanky/jdo-identita-objektu-a-dotazy-nad-databazi/

[15] Burget, R. http://interval.cz/clanky/jdo-datove-typy-a-kolekce/

[16] Burget, R. http://interval.cz/clanky/jdo-vztahy-mezi-objekty/

[17] Burget, R. http://interval.cz/clanky/jdo-prace-s-persistentnimi-objekty/

Zoznam príloh

Príloha 1. CD obsahujúce aj demo aplikáciu, ktorá používa navrhnutý objektovo-orientovaný rámec.